1. How to debugging pod in Kubernetes

Ans: kubectl get pods

kubectl describe pod <pod-name>

2.How to delete pod

1. **Delete a Pod by Name**:

kubectl delete pod <pod-name>

2. **Delete Pods Using Labels**:

kubectl delete pods -l <label-key>=<label-value>

3. **Delete Pods in a Namespace**

**kubectl delete pod -n <namespace> <pod-name>**

4. Delete All Pods in a Namespace

kubectl delete pods --all -n <namespace>

3. **How to restore pods in k8s?**

In Kubernetes, restoring pods typically involves recreating or scaling the deployment, statefulset, or replication controller that manages the pods. The exact steps may vary depending on the type of workload you are using. Here are general steps for different types of workloads:

**Deployments:**

If you're using Deployments, you can update the deployment to trigger the creation of new pods. The old pods will be terminated, and new ones will be created with the updated configuration

**StatefulSets:**

For StatefulSets, you can also update the StatefulSet to trigger pod recreation. However, StatefulSets provide stable network identities for pods, so rolling updates may be more gradual

**Replication Controllers:**

If you are using Replication Controllers (although Deployments are more commonly used), you can scale the replication controller to a desired number of replicas

kubectl scale rc your-replication-controller --replicas=3

5. **How to execute or build docker image**

1.Create docker file

# Use an official Node.js runtime as a base image

FROM node:14

# Set the working directory in the container

WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory

COPY package\*.json ./

# Install app dependencies

RUN npm install

# Copy the application files to the working directory

COPY . .

# Expose a port for the application

EXPOSE 3000

# Define the command to run the application

CMD ["npm", "start"]

2.Build the docker image

docker build -t your-image-name:tag .

3. **Verify the Built Image:**

docker images

4. **Run a Container Based on the Image:**

docker run -p 8080:3000 your-image-name:tag

**6.**Key Differences difference b/w git fetch and git pull?

* **git fetch** only downloads changes from the remote repository but does not automatically integrate them into your local branch.
* **git pull** is a combination of **git fetch** followed by an automatic merge, potentially leading to merge conflicts.
* **git fetch** is typically used when you want to review changes before integrating them or when you want to update your local repository without affecting your current branch.
* **git pull** is used when you want to quickly update your local branch with the latest changes from the remote.

7. What are azure Artifacts?

Azure Artifacts enables developers to share their code efficiently and manage all their packages from one place. With Azure Artifacts, developers can publish packages to their feeds and share it within the same team, across organizations, and even publicly.

1. **Centralized Package Management**:
   * Azure Artifacts provides a centralized location to store and manage all your organization's software artifacts and packages.
2. **Support for Multiple Package Types**:
   * It supports various package formats, including NuGet, npm, Maven, and Python packages. This allows you to manage dependencies for a wide range of projects and applications.
3. **Private Feeds for Secure Package Storage**:
   * You can create private feeds to store sensitive or proprietary packages securely within your organization. This ensures that only authorized team members can access and use these packages.

8.What is Different b/w Variable and Variable Group?

Variables:

Scope: Variables are defined at the pipeline level. They are specific to a single pipeline and can be used within that pipeline and its stages, jobs, and tasks.

Syntax :

variables:

myVariable: 'someValue'

Variable Groups:

Scope: Variable groups are defined at the project or organization level. They can be used across multiple pipelines within the same project or across projects in the organization

Syntax:

variables:

- group: 'MyVariableGroup'

AKS

9.What is different b/w **Deployments**, **StatefulSets** and **Daemonsets**?

**Deployments** are great for stateless applications that can be easily scaled horizontally**,**

**StatefulSets** are great for applications that require persistent storage and have state that needs to be maintained.

**DaemonSets** are great for running an application on every node in the cluster**.**

**Deployments:**

* **Purpose:** Deployments are a high-level abstraction that manages ReplicaSets. They are primarily used for stateless applications.
* **Scaling:** Deployments support easy scaling and rolling updates. They ensure that a specified number of replicas of the application are always running.
* **Pods:** Pods managed by Deployments are typically interchangeable, and the application is designed to be stateless.

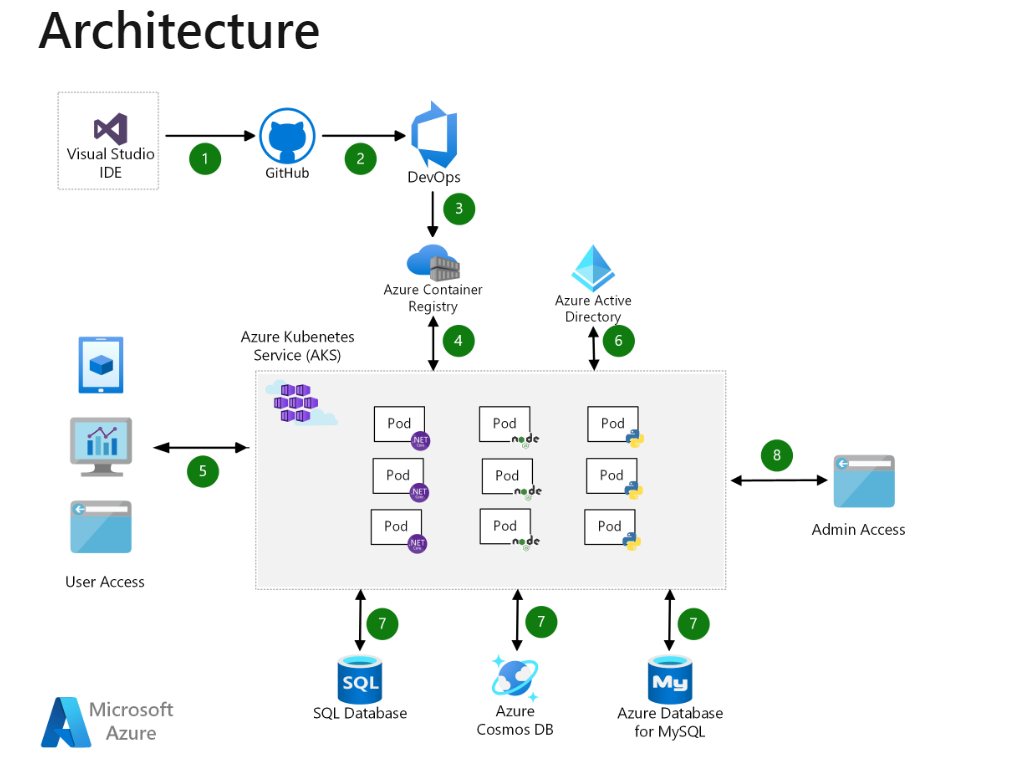
**StatefulSets:**

* **Purpose:** StatefulSets are designed for stateful applications that require stable network identifiers and stable storage. Examples include databases.
* **Pods:** StatefulSets assign a unique and persistent hostname to each pod, and they manage the order of pod creation and deletion.
* **Scaling:** Scaling with StatefulSets is typically done manually, and they are suitable for applications that require a fixed network identity.

**DaemonSets:**

* **Purpose:** DaemonSets ensure that a copy of a pod runs on each node in the cluster. They are typically used for cluster-level services like log collectors or monitoring agents.
* **Pods:** Each node runs exactly one copy of the pod managed by the DaemonSet.
* **Scaling:** DaemonSets automatically scale as nodes are added or removed from the cluster.





10. Explain Kubernetes architecture?

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. The architecture of Kubernetes is designed to be scalable, extensible, and resilient. Here's an overview of the key components in the Kubernetes architecture.

1. **Master Node:**
   * **API Server:** The API server is the central management entity that exposes the Kubernetes API. It processes RESTful API requests, validates them, executes corresponding operations, and updates the etcd datastore.
   * **Controller Manager:** The Controller Manager is responsible for regulating the state of the system. It includes various controllers that watch for changes in the cluster state (e.g., nodes, pods) and take corrective action to ensure the desired state is maintained.
   * **Scheduler:** The Scheduler is responsible for placing containers onto available nodes based on resource requirements, policies, and constraints. It makes decisions on where to deploy new pods.
   * **etcd Datastore:** etcd is a distributed key-value store that acts as the cluster's source of truth for all configuration data. It stores the desired state of the cluster and serves as the persistent storage backend for all cluster data.
2. **Worker Nodes:**
   * **Kubelet:** The Kubelet is an agent that runs on each node in the cluster. It is responsible for communicating with the API server, managing the container lifecycle (starting, stopping, restarting), and ensuring that containers are healthy.
   * **Kube Proxy:** Kube Proxy maintains network rules on nodes, allowing communication between different pods and external network entities. It performs network address translation (NAT) and load balancing for services.
3. **Pods:**
   * **Pods:** Pods are the smallest deployable units in Kubernetes. They are logical groups of one or more containers that share the same network namespace and storage. Containers within a pod communicate over the localhost, making them suitable for closely coupled processes.
4. **Service:**
   * **Service:** A Kubernetes Service is an abstraction that exposes a set of pods as a network service. It provides a stable IP address and DNS name, enabling other applications to discover and communicate with the pods.
5. **Volume:**
   * **Volume:** Volumes provide persistent storage to containers within a pod. They can be used to share data between containers, persist data beyond the lifecycle of a pod, or provide storage for stateful applications.
6. **Namespace:**
   * **Namespace:** Namespaces provide a way to create virtual clusters within a physical cluster. They help in organizing and isolating resources, allowing multiple users or teams to use the same cluster without interfering with each other.

Documentation link: <https://kubernetes.io/docs/concepts/overview/components/>

**11.What Types of Services in K8s?**

* **ClusterIP:** Exposes the Service on a cluster-internal IP address. This type is accessible only within the cluster.
* **NodePort:** Exposes the Service on each Node's IP address at a static port. This type allows external access to the Service.
* **LoadBalancer:** Creates an external load balancer in cloud environments (e.g., AWS, GCP, Azure) and assigns a stable external IP address to the Service.
* **ExternalName:** Maps the Service to the contents of the **externalName** field (e.g., DNS name).

1. What is Default Service?

Cluster IP

b. What is different NodePort and Load balancer?

**Key Differences:**

* **NodePort:**
  + Exposes the service on each node at a static port.
  + Direct access using any node's IP address and assigned NodePort.
  + Suitable for development and testing environments.
* **LoadBalancer:**
  + Uses the cloud provider's load balancer to expose the service.
  + Cloud provider assigns an external IP address that routes traffic to the service.
  + Suitable for production environments requiring a scalable and highly available external endpoint.